

# SCRIPT DE CREATION DE LA BASE DE DONNEES

Le script est intégré à l'application et est réparti entre deux modules : le Controller (get\_better\_diet et connect\_to\_mysql). Ci-dessous sont présentées des captures d'écran du code pour un accès plus facile au livrable demandé.

## CONTROLE DE L'EXISTENCE D'UNE BASE ET CREATION EVENTUELLE

```
def step_select_action(self):  
    """ ...  
  
    # Check whether a local DB has already been created.  
    while True:  
        try:  
            # Check whether a connection with an existing local DB is OK.  
            self.queries = sql.ORMConnection()  
            break  
        except Exception:  
            # Creation of a connexion, iot prepare the DB creation  
            self.itf.clear_window("right")  
            self.itf.right_display_info(cfg.WARNING_MESSAGE_4, "warning")  
            self.__create_cnx_parameters()  
            # Creation of the DB through a method hosted in this module  
            self.__initialize_DB()  
    self.itf.title_bar(cfg.TITLE_2)  
    # Display a drop down menu to navigate in the application  
    self.itf.clear_window()  
    self.itf.left_display_string(0, cfg.INFO_LINE_1)  
    self.itf.display_result(cfg.INFO_DISPLAY_RESULTS)  
    answer = self.itf.set_up_drop_down(cfg.OPERATE_ON_DB,  
                                     cfg.SELECT_ANSWER)  
    # Here start the work on the DB to find, select and record a product  
    y = 0
```

## SAISIE DES PARAMETRES DE CREATION DE LA BASE

```
def create_cnx_parameters(self):
    """ ...
    self.interface.clear_window("left")
    self.interface.display_guide(cfg.USER_GUIDE)
    y = 0
    # Ask for the connection parameters. Default value in config.py
    self.interface.left_display_string(y, cfg.DB_INITIAL_INFO)
    user, y = self.interface.display_string_textpad(1, 1, 15, ...
    user = self.ascii_to_string(user)
    if user != '':
        cfg.DB_USER = user
    password, y = self.interface.display_string_textpad(y, 1, 20,
    cfg.DB_PASSWORD_INVITE)
    password = self.ascii_to_string(password)
    if password != '':
        cfg.DB_PASSWORD = password
    connection_string = cfg.DB_CONNEXION_STRING.format(
        cfg.DB_USER, cfg.DB_PASSWORD, "")
    # Connection parameters are saved in a separate file to be reused.
    with open(cfg.DB_PARAMETERS, "w") as file:
        file.write(connection_string)
```

## CREATION D'UNE NOUVELLE BASE

```
def __initialize_DB(self):  
    """ ...  
    self.queries = sql.ORMConnection()  
    self.queries.create_database()  
    self.itf.right_display_info(cfg.DB_CREATE_LOCAL_DB)  
    self.queries = sql.ORMConnection()  
    # Open the connection to the local DB  
    self.queries.open_session()  
    # Import categories from Open Food Facts  
    OFF_categories = self.OFF.import_static_data(coff.URL_STATIC_CAT)  
    # Configure the data and upload categories into the local DB  
    self.queries.upload_categories(OFF_categories)  
    self.itf.right_display_info(cfg.DB_CATEGORIES_FETCHED)  
    # Import stores from Open Food Facts  
    OFF_stores = self.OFF.import_static_data(coff.URL_STATIC_STORES)  
    # Configure the data and upload stores into the local DB  
    self.queries.upload_stores(OFF_stores)  
    self.itf.right_display_info(cfg.DB_STORES_FETCHED)  
    # Informs the user that the DB is empty.  
    self.itf.right_display_info(cfg.EMPTY_DB, "warning")
```

## CONFIGURATION DES TABLES PARENTES

```
def create_database(self):
    """ ...

    # Create a new and empty database
    with open(cfg.DB_PARAMETERS) as file:
        connection_parameters = file.read()
    self.__engine = create_engine(connection_parameters, echo=False)
    # Activate the Database to subsequently create the tables
    connection = self.__engine.connect()
    connection.execute(COMMIT)
    connection.execute(CREATE_DB)
    connection.close()
    # Add the name of the database to the parameters file for further use
    connection_parameters = connection_parameters + cfg.DB_NAME
    with open(cfg.DB_PARAMETERS, "w") as file:
        file.write(connection_parameters)
    # Add the tables to the new database
    self.__engine = create_engine(connection_parameters, echo=False)
    self.__engine.connect()

    Base.metadata.create_all(self.__engine)
```

```
Base = declarative_base()
```

```
class Category(Base):
```

```
    """ ...
```

```
    __tablename__ = 'category'
```

```
    __table_args__ = (Index('idx_category', 'name'),)
```

```
    id_category = Column(Integer(), primary_key=True, autoincrement=True,  
| | | | nullable=False)
```

```
    name = Column(String(600), nullable=False)
```

```
class Product (Base):
```

```
    """ ...
```

```
    __tablename__ = 'product'
```

```
    code = Column(String(13), nullable=False, primary_key=True)
```

```
    brand = Column(String(200), nullable=False)
```

```
    name = Column(String(600), nullable=False)
```

```
    nutrition_grade = Column(String(1), nullable=False)
```

```
class Store (Base):
```

```
    """ ...
```

```
    __tablename__ = 'store'
```

```
    __table_args__ = (Index('idx_store', 'name'),)
```

```
    id_store = Column(Integer(), nullable=False, primary_key=True,  
| | | | autoincrement=True)
```

```
    name = Column(String(600), nullable=False)
```

# CONFIGURATION DES TABLES DE JOINTURE

```
class CategoryProduct (Base):
> """ ...
    __tablename__ = 'category_product'

    id_cat_prod = Column(Integer(), primary_key=True, autoincrement=True,
                           nullable=False)
    idcategory = Column(Integer(),
                        ForeignKey('category.id_category',
                                   name='FK_id_category'), nullable=False)
    code = Column(String(13),
                  ForeignKey('product.code', name='FK_product_category',
                              ondelete='CASCADE', onupdate='CASCADE'),
                  nullable=False, )

class StoreProduct (Base):
> """ ...
    __tablename__ = 'store_product'

    id_store_product = Column(Integer(), primary_key=True, autoincrement=True,
                               nullable=False)
    product_code = Column(String(13),
                          ForeignKey('product.code', name='FK_product_store',
                                      onupdate='CASCADE', ondelete='CASCADE'),
                          nullable=False)
    store_id = Column(Integer(),
                      ForeignKey('store.id_store', name='FK_store_id',
                                  onupdate='CASCADE',
                                  ondelete='CASCADE'), nullable=False)

class ProductComparrison (Base):
> """ ...
    __tablename__ = 'product_comparrison'

    id_prod_comp = Column(Integer(), primary_key=True, autoincrement=True,
                           nullable=False)
    code_best_prod = Column(String(13),
                            ForeignKey('product.code', name='FK_code_product_best',
                                        onupdate='CASCADE', ondelete='CASCADE'),
                            nullable=False)
    code_ref_prod = Column(String(13),
                           ForeignKey('product.code', name='FK_code_product_ref',
                                       onupdate='CASCADE', ondelete='CASCADE'),
                           nullable=False)
    date_best = Column(DateTime(), nullable=False)
```